25/03/2020

Projet Développement Web

Rapport coaching 8

Mise en place d'un site via Symfony 5 et React.js

VAN DEN DORPE Ludo
Pestiaux Tristan

Sécurité

Côté serveur

Notre site est hébergé sur un vps qui nous a été fourni dans le cadre de ce cours.

Premièrement notre site est disponible en https avec un certificat ssl non-validé (car payant) mais la sécurité https reste toutefois active.

Ensuite, nous avons configurer le framework Fail2Ban, ce qui va nous protéger des attaques contre nos services en bloquant les adresses IP inconnues qui tentent de pénétrer dans notre système.

De plus, l'accès au vps via l'utilisateur root est désactivé, pour y accéder nous utilisons un compte que nous avons créé.

Enfin, le port sftp de base(22) a été bloqué et changé (5285) donc plus difficile à trouver en cas d'attaque.

Côté logiciel

La connexion à notre site est réservée aux invités du mariage. Lors de notre première visite sur le site nous somme dirigé vers un formulaire de connexion (les identifiants de connexion seront fourni au préalable par les mariés).

Au niveau des frameworks nous utilisons Symfony (v4.14.4 soit l'une des dernière version) ce qui nous assure une certaine sécurité puisque nous utilisons les outils de ce framework essentiellement. D'abord pour le lien avec la base de donnée qui se fait via l'ORM de Symfony (Doctrine v5.0.5), ensuite pour notre api (Api-Platform v2.5) qui a été générée par une librairie Symfony, enfin nous utilisons l'outil de sécurité (symfony/security v5.0.5) de symfony pour gérer les utilisateurs et encoder leur mot de passe.

L'utilisation de l'outil de sécurité de Symfony nous permet de choisir l'encodage des mots de passes (algorithm: bcrypt cost: 12).Cela nous permet également la création de rôles (ROLE_ADMIN et ROLE_USER) afin de bloquer l'accès au site (via url) à des utilisateurs non-identifiés mais aussi de distinguer l'accessibilité de certaines pages en fonction du rôle de l'utilisateur (exemple: page administration).

```
    access_control:
        - { path: ^/admin, roles: ROLE_ADMIN }
        - { path: ^/user, roles: [ROLE_USER, ROLE_ADMIN] }

    encoders:
        App\Entity\Utilisateur:
            algorithm: bcrypt
            cost: 12
```

Côté Base de données

La première sécurité mise en place est la connexion de notre Api vers la Base de données : cette connexion ne se fait pas en root.

Deuxièmement, Nous avons créé des groupes d'utilisateurs afin d'empêcher de faire des requêtes si nous ne sommes pas connectés au site et de bloquer certaines requêtes seulement disponibles pour les administrateurs.

```
/**
 * @ApiResource(
 *  collectionOperations={
 *         "post"={"path"="/invites", "security"="is_granted('ROLE_ADMIN')"},
 *          "get"={"path"="/invites", "security"="is_granted('ROLE_USER') or is_granted('ROLE_ADMIN')"}
 *     },
 *     itemOperations={
 *         "get"={"path"="/invites/{id}", "security"="is_granted('ROLE_USER') or is_granted('ROLE_ADMIN')"},
 *         "delete"={"path"="/invites/{id}", "security"="is_granted('ROLE_ADMIN')"},
 *         "put"={"path"="/invites/{id}", "security"="is_granted('ROLE_ADMIN')"}
 *     }
 * )
 * @ORM\Entity(repositoryClass="App\Repository\InviteRepository")
 */
```

Troisièmement, la validation des champs lors de requêtes post est gérée à tous les niveaux : d'abord une vérification frontend via react, ensuite une vérification au niveau de l'api (type de donnée attendue, NotBlank/NotNull, length,...) et enfin au niveau de la base de donnée (type de donnée attendue, NotNull).

```
/**
 * @ORM\Column(type="string", length=255)
 * @Assert\NotBlank
 */
private $nom;
```